SAMURAI: A Streaming Multi-tenant Context-Management Architecture for Intelligent and Scalable Internet of Things Applications

Davy Preuveneers and Yolande Berbers
iMinds-DistriNet
Department of Computer Science, KU Leuven
Leuven, Belgium
{*davy.preuveneers, yolande.berbers*}*@cs.kuleuven.be*

***Abstract*—In the Internet of Things, heterogeneous and distributed streams of sensor events is a driver for context-aware behavior in intelligent environments. However, processing the event data usually cross-cuts the business logic of IoT applications and offering such reusable functionality as a service towards a variety of customers with different needs is often faced with scalability concerns. We present SAMURAI, a multi-tenant streaming context architecture that integrates and exposes well-known components for complex event processing, machine learning, knowledge representation, NoSQL persistence and in-memory data grids. SAMURAI pursues a twofold approach to achieve scalability: (1) distributed deployment with horizontal scalability, (2) shared resources through multi-tenancy. For the scenario used in the experimental evaluation of our architecture, the results show little overhead to support multi-tenancy, with near-linear scalability and flexible elasticity for deployment schemes with data partitioning per tenant.**

***Keywords*-context, stream mining, complex event processing, semantic enrichment, classification, scalability**

I. Introduction

With the advent of smartphones and tablets as mobile and ubiquitous computing platforms, sensor technology has become omnipresent in our surroundings. Mainstream smartphones now have sensors to observe acceleration, location, orientation, ambient lighting, sound, imagery, etc. [1]. Furthermore, emerging computing paradigms like the Internet of Things (IoT) [2] promise a continuous data growth in an open ended dynamic network infrastructure fueled by low cost wireless communication.

With an exponential growth, tapping into this data with conventional methods for faster decision making or deeper insights has become unfeasible due to the unpredictable peaks of high computational cost to collect and process data. These challenges have amplified the need for context-aware computing solutions offered as services that can deal with:

1) Customers sharing the same services and infrastructure
2) Heterogeneous data sources and event types
3) Loosely structured and distributed event streams

In our previous work [3], we used smartphone accelerometer data for step counting, sleep analysis, fall detection and classifying different types of motion. With accelerometers capable of producing 100 or more updates per second, the amount of data is orders of magnitude higher compared to a temperature sensor sampled every 30 seconds. By adding complexity, such as machine learning and semantic reasoning techniques, we can further improve the recognition accuracy. However, such complexity is usually too much to handle for a smartphone. Even for server-side implementations scalability remains a concern when multiple customers with different needs must be served at the same time.

We are tackling these challenges within the frame of the FP7 BUTLER project[1] by means of a secure and context-aware architecture for the Internet of Things. The objective of BUTLER is the creation of a horizontal IoT platform supporting several domains of our daily lives – including home, health, smart cities, energy, transport, shopping, etc. – all at once. A variety of processing components have been implemented in a modular way to enable a flexible deployment composition on the following platforms:

- *Smart Object*: Small appliances, sensors or actuators with limited computational power, storage capacity, communication capability, energy supply and primitive user interface (e.g. RFID tagged objects, motion detectors, heating regulators).
- *Smart Mobile*: Devices with multi-modal user interfaces to enable user mobility through remote services (e.g. smart phones, smart TVs).
- *Smart Server*: The aggregation and complex analysis of data from smart objects and smart mobiles are realized as services on smart servers (e.g. a local server or cloud computing set-up).

In this work, we present and evaluate SAMURAI, a stream mining context architecture with components for complex event processing, machine learning, knowledge representation, NoSQL persistence and in-memory data grids. SAMURAI exposes them as RESTful services and pursues a twofold approach to achieve scalability:

1) Distributed deployment with horizontal scalability
2) Sharing resources through multi-tenancy

After reviewing related work in section II, we present a health related use case with activity recognition in section III

[1] http://www.iot-butler.eu/

as a motivating example. Section IV discusses the primitives used in SAMURAI. In section V we evaluate the feasibility and effectiveness of the approach. We conclude in section VI summarizing the main insights and identifying possible topics for future work.

II. Related work

The domain of activity recognition is rife with examples that use accelerometers [4], [5], [6] to assess physical activity. These works focus on algorithmic and technological aspects on how to recognize activities of interest. Especially in the area of home-care applications and Ambient Assisted Living [7] for the elderly, automatic discovery and classification of daily activities plays a key role [8], [9] to anticipate the kind of assistance they need or to detect the occurrence of abnormal events (such as a fall or heart failure). Many works in this field rely on numerous sensors to enrich observation data, and often depend on prior knowledge about the activities and the environment learned in a supervised manner. However, the domain specific solutions are often restrained within the boundaries of the home environment and as such do not lend themselves that well to learn different types of activities on a larger scale.

When processing large amounts of data, the link with the *Big Data* paradigm is not far off. To achieve faster results with parallel and distributed computing, MapReduce [10] and its open source implementation Hadoop[2] are currently the standards in this area. However, in our ecosystem where most data is generated in a streaming fashion, the batch-oriented approach of the MapReduce programming paradigm is fundamentally different from the streaming oriented processing that we need to deal with the velocity of the data in (near) realtime.

Stream mining [11] differs from more traditional data mining techniques and tools like Weka [12] in the sense that they focus on extracting knowledge in non-stopping streams of events. Massive Online Analysis (MOA) [13] is such a software framework that offers a variety of algorithms and evaluation methods for supervised and unsupervised learning, supporting only single machine deployments thereby limiting its scalability. Mahout[3], a machine learning and data mining framework built on top of Hadoop, addresses this scalability challenge. However, due to the Hadoop dependency, Mahout is batch-oriented rather than streaming oriented. Other distributed stream processing engines include S4 [14] and the Store project[4] though they put less emphasis on stream mining as a core activity.

With SAMURAI, we aim to fill this gap by offering a *distributed* and *multi-tenant* event-based *streaming architecture* with *data mining* and *semantic enrichment* capabilities. A running instance of SAMURAI and additional documentation is available at *https://butler.cs.kuleuven.be/samurai*.

[2]http://hadoop.apache.org
[3]http://mahout.apache.org
[4]http://storm-project.net

III. Motivating use case

In this section we briefly discuss a motivating use case building upon earlier work carried out in the frame of a mobile application for diabetes patients [15], [16]. The major challenge in this application is to identify classes of activities that have an effect on blood glucose levels. As *activities of daily living* (ADL) typically present recurring behavioral patterns, we explore correlations between *time* and *location* on the one hand and types of activities on the other hand, to find similar situations of the past as a recommendation for the patient. We also track the number of steps taken each day as a measure for well-being and as a means for recommendation to have a more active lifestyle.

As a running example, we will demonstrate how SAMURAI processes multiple event streams and explores spatio-temporal characteristics among co-occurrent events to probabilistically classify the activity of an individual:

- Accelerometer events for motion activity recognition
- SSID and RSSI events of an active WiFi connection
- GSM Cell ID and Location Area Code events

This use case presents various technical challenges for a multi-user deployment:

- **Feature extraction:** Convert raw accelerometer data into meaningful features (e.g. walking, running, number of steps, activity intensity)
- **Information fusion:** Aggregate data from different sources to increase the confidence in the quality of the inferred information (e.g. current activity w.r.t. to current time and location)
- **Domain knowledge:** Leverage background information to narrow down likely activities (e.g. semantically linking locations with activity types)
- **Probabilistic correlations:** Identify frequent co-occurrences in event streams to derive event patterns of interest (e.g. spatio-temporal patterns)

In the following section, we will discuss the basic primitives that we use in our framework to address these challenges.

IV. Event-based streaming context management and processing

Our event-based streaming architecture uses the Spring Framework[5] and Jersey[6] to expose well-known software libraries for complex event processing, machine learning and knowledge representation as RESTful services. These are illustrated in Figure 1.

[5]http://projects.spring.io/spring-framework/
[6]https://jersey.java.net/

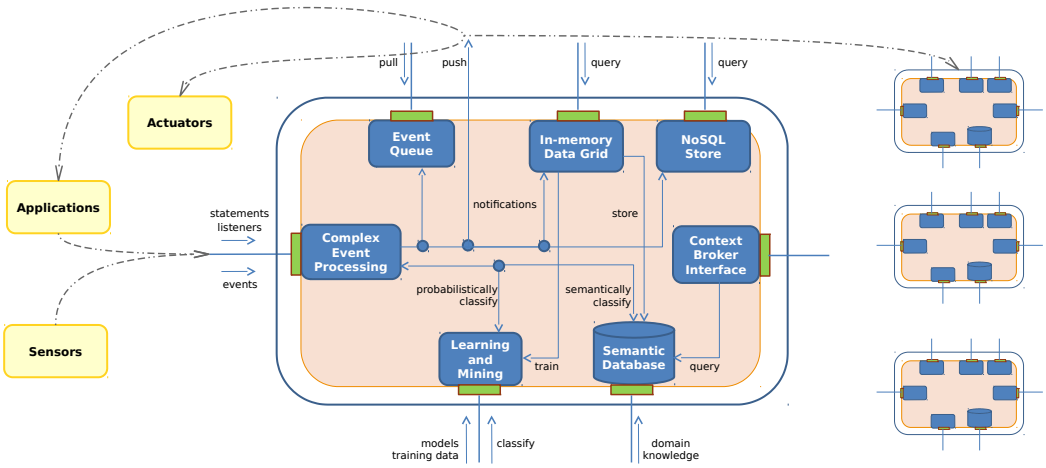

Figure 1. Conceptual overview of streaming architecture

A. Basic principles and building blocks

Events can be *simple events* that carry slivers of meaning in themselves, and *complex events* which summarize, represent, or denote a set of single events which combined denotes a pattern of events. An event is represented as a set of typed key-value pairs that can be easily serialized into the JSON format. The example below illustrates the type and an instance of an accelerometer event that we use for activity recognition. In this example, the *x*, *y* and *z* values hold the acceleration values along these axes. The *timestamp* field represents the number of milliseconds passed since January 1, 1970 UTC:

```
{                                    {
  "timestamp": "long",                 "timestamp":1340099550210,
  "x":"double",                        "x":-8.308,
  "y":"double",                        "y":-1.9477,
  "z":"double"                         "z":4.099
}                                    }
```

Figure 2. Example of an event type and an event instance

The architecture offers publish/subscribe capabilities to have clients (applications or subsystems) notified when particular (patterns of) events occur with *push* notifications implemented as REST callbacks (see following section). The architecture has three basic components to hold events:

- **In-memory Data Grid**: This is a distributed in-memory container for events based on Hazelcast[7].
- **Event Queue**: Clients that do not support push notifications through REST callbacks can register a queue to hold events and poll that instead.
- **NoSQL Store**: The events can be optionally stored in a persistent way using CouchDB[8] as a RESTful database. Their RESTful APIs follow the CRUD mapping on HTTP methods to create (POST), read (GET), update (PUT) or delete (DELETE) events.

B. Complex event processing

For step counting and fall detection, we process events from the tri-axial accelerometer found in most smartphones. Below is a short overview of the domain specific event stream processing algorithms used in our use case:

- **Accelerometer**: It produces a continuous stream of X,Y,Z acceleration data at a certain rate (e.g. 100Hz).
- **Low-pass filter**: We use the 'moving average' to remove high-frequency noise to track steps as acceleration peaks at a frequency of max 5 steps per second.
- **Magnitude filter**: We carry out the signal analysis on the magnitude of the acceleration signal as the sensor orientation may change while moving around.
- **Peak filter**: This component extracts maxima and minima in the time domain. A single step is characterized by a particular pattern of these features.
- **Step detector**: It identifies the correct maxima/minima to correctly count the number of steps and to differentiate between standing still, walking and running.
- **High-pass filter**: This component implements a FIR filter to detect sudden and high-frequency changes of the acceleration signal for fall detection.
- **Fall detector**: This component analyzes the signal magnitude area (SMA) of the high-frequency part of the acceleration signal, and identifies a fall if this feature passes a certain threshold.

A more detailed discussion of the algorithms can be found in our Intelligent Environments 2013 award winning work [17]. SAMURAI uses Esper[9] for on-the-fly processing of complex event streams. Esper enables:

[7]http://www.hazelcast.com/
[8]http://couchdb.apache.org/
[9]http://esper.codehaus.org

```
1  curl -X POST --data '{ "timestamp": "long", "x": "double", "y": "double", "z": "double" }'
2      http://localhost/samurai/rest/esper/eventtypes/AccelerometerEvent
3
4  curl -X POST --data '{ "type": "AccelerometerEvent", "timestamp": 1234, "x": 5.0, "y": 1.3, "z": 2.1 }'
5      http://localhost/samurai/rest/esper/event
6
7  curl -X POST --data '{ "rule":"insert into MagnitudeEvent(timestamp, magnitude) select timestamp, Math.sqrt(x*x + y*y + z*z) as magnitude
8      from AccelerometerEvent" }' http://localhost/samurai/rest/esper/statements/magnitude
9
10 curl -X POST --data '{ "rule":"insert into MovingAverageEvent(timestamp, movingaverage) select timestamp, avg(magnitude) as movingaverage
11     from MagnitudeEvent.win:length(10)" }' http://localhost/samurai/rest/esper/statements/movingaverage
12
13 curl -X POST --data '{"url":"http://otherhost/myapp/steps/offer"}' http://localhost/samurai/rest/esper/statements/steps/listener
```

Figure 3. Registering event types (lines 1-2), sending events (lines 4-5), registering event statements (lines 7-11), registering event listeners (line 13)

```
1  ex:Room            a                owl:Class;
2                     rdfs:subClassOf  geo:Feature .
3
4  ex:LivingRoom      a                ex:Room;
5                     rdfs:label       "Living Room";
6                     geo:hasGeometry  ex:GeoLivingRoom .
7
8  ex:GeoLivingRoom   a                sf:Polygon;
9                     geo:asWKT        "POLYGON ((0.00 9.44,3.80 9.44,
10 3.80 8.13,8.00 8.13,8.00 13.90,0.00 13.90,0.00 9.44))"^^sf:wktLiteral .
11
12 ex:activity        a                owl:DatatypeProperty;
13                    rdfs:domain      ex:Room;
14                    rdfs:range       xsd:string .
15
16 ex:LivingRoom      ex:activity      "Watch TV" .
17 ex:LivingRoom      ex:activity      "Listen to music" .
18 ex:LivingRoom      ex:activity      "Play game" .
19 ex:LivingRoom      ex:activity      "Read newspaper" .
```

Figure 4. Semantic representation of rooms and activities in an apartment

- Feature extraction from low-level events (e.g. from accelerometer to steps)
- Publish/subscribe interaction with applications or SAMURAI subsystems.

Esper usually relies on Java POJOs to represent events at compile time. However, in our IoT ecosystem new event types can be created anytime. We therefore expose a RESTful API to dynamically register new event types at runtime. Figure 3 illustrates how to do this with *curl*, a command-line utility commonly found on Linux systems to transfer data from or to a server. Registering the other event types and sending events is done in a similar way as shown in the same figure.

Our architecture offers RESTful APIs to register *statements* and *listeners*. A statement is a continuous query registered with an Esper engine instance that provides results to listeners as new events arrive. In order for applications or subsystems to be notified about the *step* events, we add a listener to this statement as shown in line 13. The example adds a REST callback to *http://otherhost/myapp/steps/offer*, which gets called upon using a HTTP GET request for every step event. The event attributes are appended to the REST callback as url parameters. This way, the *myapp* subscriber is notified about all the event details.

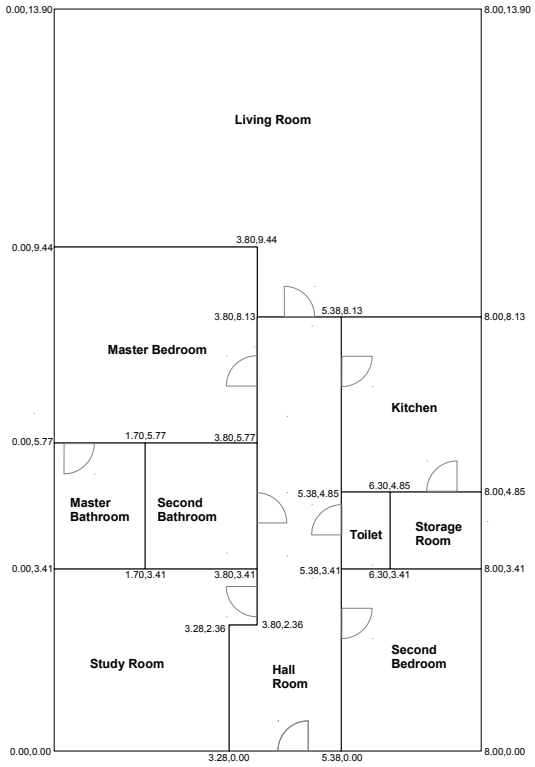

Figure 5. Visualization of the apartment

C. Semantic database with spatio-temporal reasoning

Beyond matching patterns of events and feature extraction, SAMURAI can also leverage background knowledge stored in a semantic database to increase the meaningfulness of an event. For semantic and spatio-temporal reasoning, SAMURAI uses a GeoSPARQL enabled storage backend (e.g. Parliament[10]). The benefits are manifold:

- Describe the spatial characteristics of different locations in your environment (see Figures 4 and 5).
- Use the W3C SSN ontology to describe the sensors and their position
- Translate positions in coordinates into semantic locations (e.g. [6.0, 10.0] being in the *Living Room*)

[10]http://parliament.semwebcentral.org/

```
1 curl -X POST --data '{ "classifier": "weka.classifiers.bayes.NaiveBayesUpdateable" }' http://localhost/samurai/rest/weka/models/m01
2
3 curl -X POST --data-binary @m01.arff http://localhost/samurai/rest/weka/models/m01/arff
4
5 { "rule" : "select time, x, y, classify('m01', hour(time), location(x,y), '?') from ..." }
```

Figure 6. Registering a classifier (line 1), upload training data (line 3), custom Esper operator for classification (lines 5)

- Semantically link locations with activities (e.g. *Watch TV* in a *Living Room*)

The following (simplified) statement demonstrates the integration with Esper (see Figure 7):

```
{ "rule" : "select x,y,location(x,y) from LocationEvent" }
```

Figure 7. Custom geo-semantically enhanced event operator *location()*

This statement translates the x and y coordinates (e.g. obtained after signal strength triangulation) of incoming events of type *LocationEvent* with the custom *location()* Esper operator offered by SAMURAI. The operator is mapped onto a GeoSPARQL query which retrieves the semantic location (e.g. *location(6,10)* $\rightarrow$ *'Living Room'*). Such higher level concepts are more suitable for classification.

D. Learning and mining with classification and clustering

When the relationship between co-occurrent events cannot be established in advance, we need classification and clustering mechanisms to probabilistically infer these dependencies. SAMURAI embeds the Weka [12] machine learning library for this purpose and exposes its key features through RESTful APIs. SAMURAI allows every application to register one or more *models*, with each model having a particular attribute set and classifier. See line 1 in Figure 6. This example registers a model called *m01* using Naive Bayes as an incremental classifier. The attributes used for classification are described in the Attribute-Relation File Format (ARFF) and registered with the following REST API (see line 3). By specifying an appropriate statement and corresponding listener, Esper feeds events as training or test instances into the Weka model. The example in line 5 illustrates how to probabilistically classify activities from the current time (in hours) and location (e.g. *8,'Kitchen'* $\rightarrow$ *'HavingBreakfast'*). This example demonstrates the use of Weka to learn spatio-temporal correlations. The integration with Esper is again with custom Esper operations mapping the core classification and clustering features of Weka. Many technical details of the RESTful APIs and examples could not be elaborated upon in depth in the previous sections due to lack of space. These will be offered on the website *https://butler.cs.kuleuven.be/samurai/* that hosts a running instance of SAMURAI.

E. Multi-tenancy and data isolation per tenant

The Software as a Service (SaaS) / Platform as a Service (PaaS) and multi-tenancy paradigms of cloud computing are often positioned as practical approaches to offer the above functionality to a variety of customers with different needs. However, there are several concerns from a data management standpoint that make this endeavour not straightforward for context management.

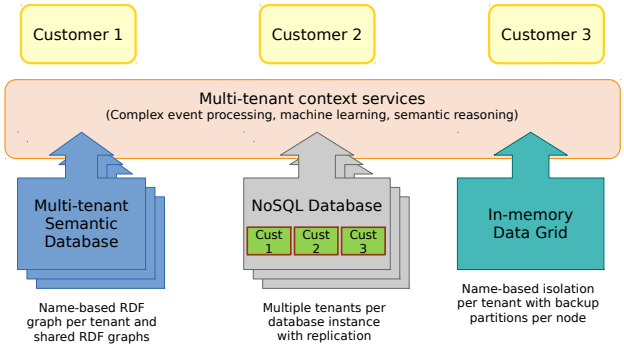

Figure 8. The multi-tenant data architecture of SAMURAI

- **Data isolation**: Separate each customer's data and context to reduce the risk of exposing the wrong data
- **Performance**: Manage the customer's data that allows for collocation or isolation based on service level agreements or performance
- **Management**: Add new customers in a flexible way and allow for cross-customer context support

In the SAMURAI architecture, data is provisioned in the *In-memory Data Grid*, the *Semantic Database*, the *NoSQL Store* and the *Event Queues* hosted on top of the in-memory data grid. Using dedicated servers per tenant offers good isolation from a security perspective, but wastes resources when the context management services are not used. Enabling multiple tenants to share a database and isolating tenant data by using separate tables for each tenant helps to reduce per tenant costs.

SAMURAI achieves multi-tenancy by having each customer authenticate and with a tenant identifier in an extra HTTP header. Access based on their identity to the persistent and in-memory data is then achieved for the 3 aforementioned data subsystems as illustrated in Figure 8. To simplify isolation per tenant in the semantic database, we instrument SPARQL queries so that the tenant only accesses its own SPARQL graph. The NoSQL store is set up in replication mode to ensure high availability. The in-memory data grid acts as a distributed hash map and is the most frequently used and highest performing data access component.

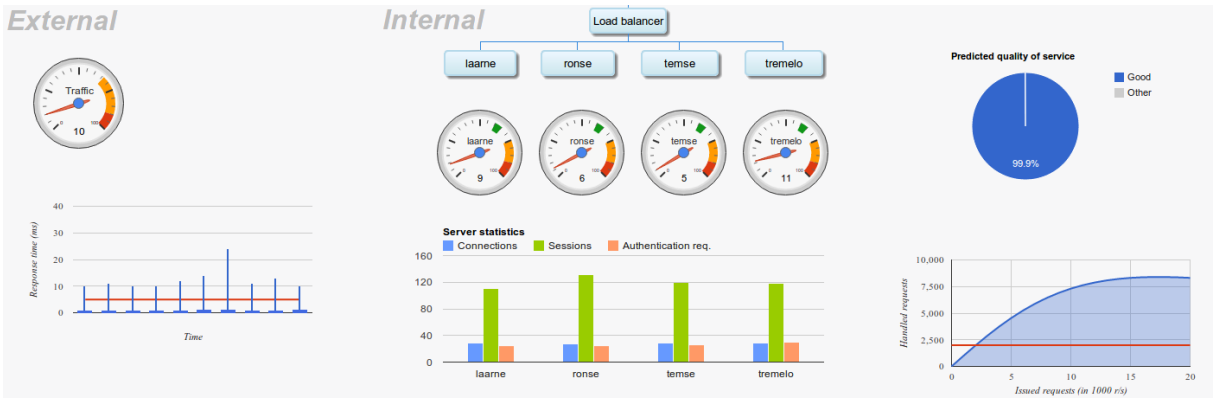

Figure 9. Monitoring dashboard

Delay/user req	Time/user req	Requests/second	CPU load
50 ms	1.29 ms	1950	± 21%
20 ms	1.22 ms	4713	± 52%
14 ms	1.66 ms	6387	± 71%
10 ms	1.81 ms	8467	± 82%
8 ms	1.95 ms	10050	± 92%
5 ms	4.08 ms	11013	± 99%
3 ms	6.09 ms	11001	± 99%

Table I
MEASURING THE PROCESSING TIME AND CPU LOAD FOR DIFFERENT WORKLOADS

V. PERFORMANCE AND SCALABILITY EVALUATION

To evaluate the running example of section III, the objective is to do activity recognition based on accelerometer event streams, spatiotemporal classification with semantic localization. The objective of the evaluation is not to evaluate the effectiveness of the recognition, but the scalability of the approach for a large user base. We will therefore evaluate in this section several architectural tactics to ensure the scalability of the SAMURAI framework in a distributed and multi-tenant deployment.

A. Experimental setup

We use an experimental setup of 10 machines, each equipped with an Intel Core 2 Duo 3.00 GHz CPU and 4GB of memory. All machines are linked to a 1 Gigabit network. We use an additional 5 machines to simulate event streams of a large user base. We refer to the former 10 machines as the *internal* side of the setup, whereas the 5 machines acting as load generators being the *external* side of the setup. Figure 9 illustrates our monitoring dashboard with SAMURAI being deployed on 4 systems (called *laarne*, *ronse*, *temse* and tremelo).

Each SAMURAI instance is deployed on an Apache Tomcat 7.0.41 application server on a 64-bit Ubuntu 12.04 system. Per user (or tenant), SAMURAI processes per second on average:

- 60 REST requests from event sources (accelerometer and location events)
- 200 events (accelerometer, location, steps and all intermediate events)
- 15 internal semantic localization and activity requests
- 15 internal probabilistic classification requests

To create a better understanding of the performance impact of the event processing itself, we measured the overhead of handling REST requests. We exposed an empty dummy method with a POST interface and simulated 100 users, each sending out REST requests to this interface. Table 1 shows the the processing time for handling REST requests and the CPU load. For a delay of 20 ms between each REST request, the processing time is 1.22 ms. The server processes 100 x (1000 / (20 + 1.22)) or about 4713 REST requests per second for all 100 users together, causing a CPU load of 52% for a REST method that required no further internal processing. This means that for every 1000 requests/second, the CPU load for pure REST processing goes up with about 10%. The maximum capacity is reached with a CPU load at 100% when the sum of the delay and the processing time per user request is about 9ms.

B. Runtime overhead of a multi-tenant setup

After tenants have authenticated, the user identity is used to instrument each SPARQL query and REST request so that the tenant accesses its own data. The actual complex event processing, machine learning, and semantic reasoning services are shared across all tenants.

In a first experiment we measured the runtime overhead between two instances where each tenant has its own dedicated instance where this instrumentation is hence not needed, and a setup where the resources are shared with customization per tenant. In the latter, the tenant has to authenticate with an identity management system for which we use OpenAM[11]. OpenAM has RESTful interfaces to

[11] http://http://openam.forgerock.org/

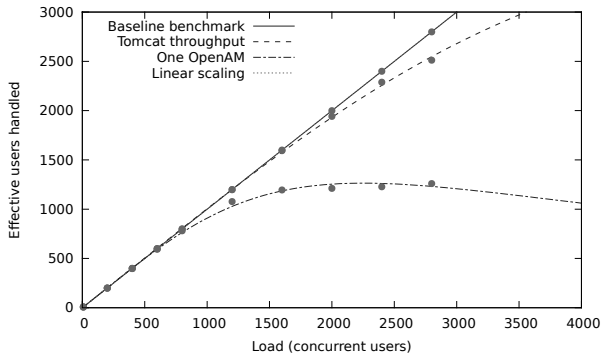

Figure 10. Measuring the overhead of Tomcat and OpenAM authentication token verification on a single server

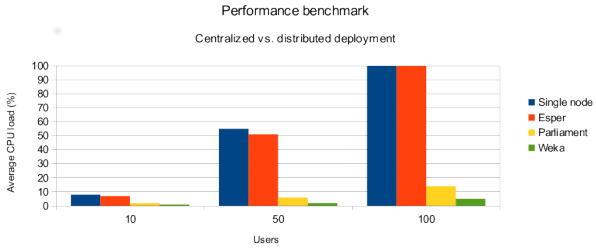

Figure 11. Centralized vs. distributed deployment of the SAMURAI subsystems

login and to verify authentication tokens in subsequent requests.

The biggest impact is during the first interaction of the tenant with SAMURAI during which the following additional steps are taking place:

- The tenant authenticates and receives a bearer token
- The tenant details are retrieved from the identify management system

Under a reasonable load, these two steps cause a latency of less than 100ms during the first request. After that, the tenant details are cached, and each additional request only requires verification of the bearer token. This simple step can be carried out in less than 5ms. The results are shown in 10. It compares 1) the impact of the load generator throughput, 2) the effect of a simple HTTP HEAD request against the Tomcat application container (without processing), and 3) the performance impact for verifying authentication tokens against a single OpenAM deployment.

C. Centralized versus distributed deployment

The subsystems in SAMURAI expose their functionality through RESTful APIs, which makes a distributed deployment of every subsystem on a different node straightforward. The first experiment compares a deployment of all subsystems on a single node with a distributed deployment having every subsystem (Esper, Parliament, Weka) on a dedicated node. Fig. 11 shows the performance results for 10, 50 and 100 users for the running example of section III.

The *Single node* bar depicts the CPU load on the centralized deployment, whereas the *Esper, Parliament* and *Weka* bars depict the CPU load on each dedicated node in the distributed deployment. One notices that for this particular experiment, the biggest impact is caused by the Esper component due to the number of events it has to process per user. With 100 users, the computational overhead of the event processing is too high for the events to be processed in (near) realtime, even when Esper runs on a dedicated host.

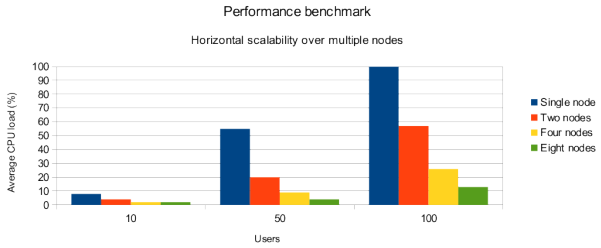

Figure 12. Horizontal scalability of the SAMURAI subsystems

D. Distributed deployment with load balancing

In this experiment, we deployed multiple instances of a complete SAMURAI system onto different nodes, and used a simple load balancer with sticky sessions to always redirect consecutive calls of clients to the same SAMURAI instance. In the background, a replication process ensures eventual consistency of the data. The performance results are shown in Fig. 12. The results show that the CPU load goes down more than linearly with the number of SAMURAI nodes, which can be explained by the fact that grouping of events per user over time takes less time with fewer users per node.

VI. Conclusion

We presented and evaluated SAMURAI, our scalable and event-based stream mining architecture that integrates and exposes software building blocks for complex event processing (feature extraction, information fusion, notification), machine learning (learn co-occurrences of events and spatio-temporal correlations) and knowledge representation (linking positions with semantic locations and activities) as RESTful services. Each of these building blocks is integrated in a loosely coupled fashion, allowing easy deployment of multiple instances of the architecture in the cloud. By mapping external functionality onto custom Esper operators that developers can use in their statements, SAMURAI is indeed a very flexible and extensible framework.

We have evaluated several architectural tactics to ensure the scalability of the SAMURAI framework in a distributed multi-tenant deployment. We carried out experiments to analyze the performance impact of pure REST request handling and the impact of multi-tenancy for data isolation.

For the scenario used in the experimental evaluation of our architecture, the results show little overhead to support multi-tenancy, with near-linear scalability and flexible elasticity for deployment schemes with in-memory data partitioning per tenant. A limitation of our current experiments was that the load generation for simulation (external) and the multi-tenant distributed deployment of SAMURAI (internal) were all linked to the same local network. As our data tier consists of multiple components (e.g. a semantic database, an in-memory data grid and a NoSQL store) each with their own replication and partitioning techniques, we believe better results can be achieved to isolate further the internal and external parts of our setup, not only for the computational part but also for the network part. Further work will focus on additional evaluations with such use cases.

ACKNOWLEDGMENTS

This research is partially funded by the Research Fund KU Leuven.

REFERENCES

[1] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *Comm. Mag.*, vol. 48, no. 9, pp. 140–150, Sep. 2010. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2010.5560598

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.05.010

[3] A. Ramakrishnan, N. Naqvi, Z. Bhatti, D. Preuveneers, and Y. Berbers, "Learning deployment trade-offs for self-optimization of internet of things applications," in *Proceedings of the 10th International Conference on Autonomic Computing, ICAC 2013*. ACM, June 2013, pp. 213–224. [Online]. Available: https://lirias.kuleuven.be/handle/123456789/411273

[4] N. Ravi, N. Dandekar, P. Mysore, and M. Littman, "Activity recognition from accelerometer data," in *Proceedings of the 17th conference on Innovative applications of artificial intelligence - Volume 3*, ser. IAAI'05, 2005, pp. 1541–1546.

[5] L. Sun, D. Zhang, B. Li, B. Guo, and S. Li, "Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations," in *Proceedings of the 7th international conference on Ubiquitous intelligence and computing*, ser. UIC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 548–562. [Online]. Available: http://dl.acm.org/citation.cfm?id=1929661.1929712

[6] J. Kwapisz, G. Weiss, and S. Moore, "Activity recognition using cell phone accelerometers," *SIGKDD Explor. Newsl.*, vol. 12, no. 2, pp. 74–82, Mar. 2011.

[7] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier, "The Internet of Things for Ambient Assisted Living," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, 2010, pp. 804–809.

[8] N. Zouba, F. Bremond, and M. Thonnat, "Multisensor fusion for monitoring elderly activities at home," in *Advanced Video and Signal Based Surveillance, 2009. AVSS '09. Sixth IEEE International Conference on*, 2009, pp. 98–103.

[9] C. Scanaill, S. Carew, P. Barralon, N. Noury, D. Lyons, and G. Lyons, "A review of approaches to mobility telemonitoring of the elderly in their living environment," *Annals of Biomedical Engineering*, vol. 34, no. 4, pp. 547–563, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10439-005-9068-2

[10] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[11] M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review," *SIGMOD Rec.*, vol. 34, no. 2, pp. 18–26, Jun. 2005. [Online]. Available: http://doi.acm.org/10.1145/1083784.1083789

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[13] P. Kranen, H. Kremer, T. Jansen, T. Seidl, A. Bifet, G. Holmes, B. Pfahringer, and J. Read, "Stream data mining using the moa framework," in *Database Systems for Advanced Applications*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7239, pp. 309–313. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29035-0_27

[14] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed Stream Computing Platform," in *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, ser. ICDMW '10, Washington, DC, USA, 2010, pp. 170–177.

[15] D. Preuveneers and Y. Berbers, "Mobile phones assisting with health self-care: a diabetes case study," in *Mobile HCI*, ser. ACM International Conference Proceeding Series. ACM, 2008, pp. 177–186.

[16] D. Preuveneers, Y. Berbers, and W. Joosen, "The future of mobile e-health application development: exploring HTML5 for a context-aware diabetes monitoring assistant," in *3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare*, October 2013 (to appear).

[17] S. N. Z. Naqvi, A. Ramakrishnan, D. Preuveneers, and Y. Berbers, "Walking in the clouds: deployment and performance trade-offs of smart mobile applications for intelligent environments," in *Proceedings of the 9th International Conference on Intelligent Environments (IE13),*, J. C. August, V. Bourdakis, D. Braga, S. Egerton, K. Fujinami, G. Hunter, F. Kawsar, A. Lotfi, D. Preuveneers, A. W. Bin Abdul Rahman, and V. Zamudio, Eds. IEEE Computer Society, Jul. 2013, pp. 212–219.